Microservice-based cloud robotics system for intelligent space

Xia, C., Zhang, Y., Wang, L., Coleman, S., & Liu, Y. (2018). Microservice-based cloud robotics system for intelligent space. *Robotics and Autonomous Systems*, *110*, 139-150. https://doi.org/10.1016/j.robot.2018.10.001

Link to publication record in Ulster University Research Portal

Published in:
Robotics and Autonomous Systems

Publication Status:
Published (in print/issue): 31/12/2018

DOI:
10.1016/j.robot.2018.10.001

Document Version
Author Accepted version

General rights
Copyright for the publications made accessible via Ulster University's Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Ulster University's institutional repository that provides access to Ulster's research outputs. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact pure-support@ulster.ac.uk.

Download date: 04/05/2023

Microservice-based Cloud Robotics System for Intelligent Space☆

Chongkun Xia[1], Yunzhou Zhang[1]*, Lei Wang[1], Sonya Coleman[2], Yanbo Liu[1]
1. College of Information Science and Engineering, Northeastern University, Shenyang, China.
2. Intelligent Systems Research Centre, University of Ulster, Derry, UK

***Abstract*— Cloud robotics (CR) is a red-hot branch of the burgeoning field of service robots that is centered on the benefits of integrating infrastructure and shared services via a cloud computing environment. Although it extends the computation power and information sharing capabilities of the network robots, the development and operations (DevOps) of the CR system are currently limited for enterprise-scale projects due to the heavy framework. In fact, current developed CR systems are typical distributed monomer architectures followed by a "top-down" design. As the scale of the applications gets larger, the operation and maintenance of CR systems will become a very difficult task. In this paper, a new architecture for a microservice-based cloud robotics system in intelligent space is proposed to solve the present dilemma. To enable this, we design a service management architecture based on a microservice to provide a highly efficient and flexible development/deployment mechanism. The container technology based on the docker engine is then used to functionally decompose the application into a set of collaborating services to ensure the software design methods, based on microservice, easy for implementation. Finally, a real experiment on SLAM (Simulation localization and mapping) in an intelligent space is implemented to verify the proposed architecture. Compared with traditional monomer architectures, the results show that the proposed framework is more productive, flexible and cost effective.**

Keywords*— Cloud robotics; microservice; container technology; cloud computing; intelligent space; visual SLAM

I. INTRODUCTION

Intelligent systems that mimic the behaviors and cognitive processes of human are rapidly being developed around the world. With the rapid development of sensor devices, the volume and type of information and data that need to be processed by the onboard processors of robots are growing rapidly. As a unique device, the robot carries out all the computation and storage processes on board, which significantly increases its computational burden and can become bloated and inefficient. To solve this problem, James J. Kuffner proposed the concept of "cloud robotics" [1]. This concept introduced a new scenario where robots were regarded as agents, relying on remote servers for most of their computational load and data storage, and creating a middleware where they can share information and knowledge. The typical structure diagram for a cloud robotics system is depicted in Fig.1. The use of cloud computing for robotics and automation brings many potential benefits such as largely ameliorating the performance of robotic systems. Since the on-board processing capacity and storage capacity are very limited for physical robots, it often leads that robots have a long processing time and run slowly. Cloud robotics [2] can not only solve the inherent problems of traditional robotic systems, such as onboard computation and storage limitation, asynchronous communication and compatibility problem of multi-robot systems, but can also enhance the performance via concepts such as a remote brain, shared knowledge-base, collective learning and intelligent behavior.

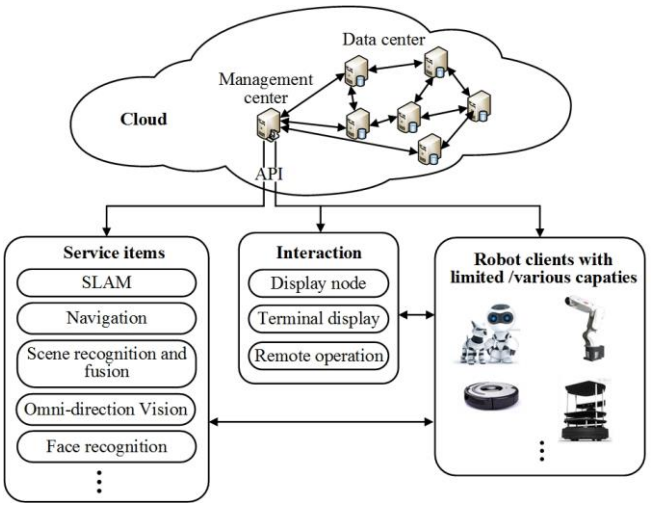


Fig.1 The cloud robotics system provides a management center and a data center. Every robot service can be registered on the cloud servers by a uniform interface standard and rule. The robot clients are the low-cost robot platforms with an embedded-class processor and a wireless connection. Robot clients can request these robot services from a service item which is stored in the management center.

The standard CR system follows a systematic "top-down design", which can be treated as a stepwise design or a synonym of decomposition. The approach in [2] describes an overview of the system, specifying, but not detailing any first-level subsystems. Moreover, as the scale of the applications expands, it may be very difficult to deploy and maintain the system for a long time [3]. In fact, current research into the CR framework is based on a typical distributed monomer architecture [4], such as M2M/M2C (Machine-to-machine/machine-to-cloud) [5] or UNR-PF (Ubiquitous Network Robot Platform) [6]. It should be noted that the monomer architecture is a mainstream development framework in the current software development background due to wide popularity, friendly IDE (Integrated Development Environment) and facilitated resource-sharing functionality. But the monomer architecture is limited by a technology stack, which forces developers to use a unified

☆Research supported by National Natural Science Foundation of China (No.61471110, 61733003), National Key R&D Program of China (No.2017YFC0805000/5005), Fundamental Research Funds for the Central Universities (N160413002, N172608005). Chong-kun Xia is the first author; he is a Ph.D. candidate. E-mail: xiachongkun@163.com.

*Corresponding author: Prof. Yun-Zhou Zhang; Tel: +86-24-83687761, e-mail: zhangyunzhou@mail.neu.edu.cn.

programming language, even though that may be inappropriate. Besides, too much coupling between services worsens the problems caused by code duplication for the monomer architecture. Hence, even though M2M/M2C and UNR-PF are widely known in academia, they cannot get out of the laboratory and achieve popularity and recognition from the market. These existing problems indicate that current architectures are unsuitable and unfavorable for the long-term development of cloud robotics system. Given this, the microservice, as a new software design idea, provides a novel perspective for technology companies and developers.

Therefore, inspired by the new design idea, in this paper we present a new cloud robotics system framework based on a microservice that tries to meet the requirements of designers and developers in an intelligent environment. Our idea is that all functions of the CR system can be modularized and regarded as a service, and some complicated tasks can be effectuated through the composition of available services. Finally, a real experiment "3D Visual Mapping and Localization" in an intelligent environment is adopted to verify the new architecture. We also design a comparative experiment to demonstrate the excellent performance of our proposed architecture. The results show the promise of our work.

II. Related work

A. Cloud-enabled robotics

Cloud technology-based computing or simply Cloud Computing is one of the most active fields of Info-Communication Technologies (ICT) [7]. The principal structure of cloud computing is depicted in Fig.2.

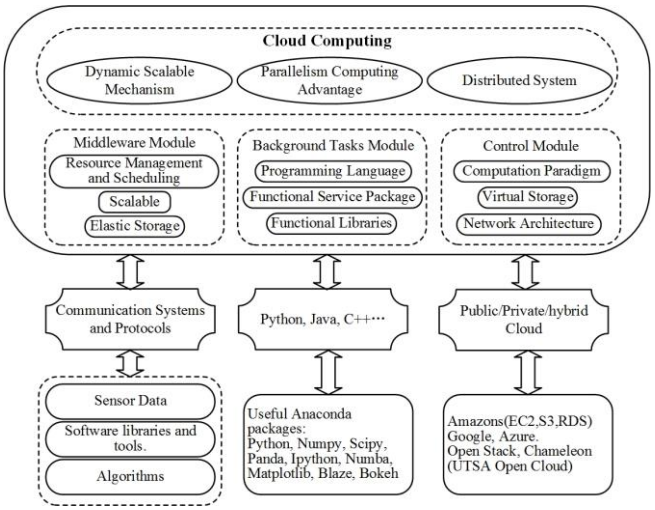


Fig.2 The cloud computing system consists of middleware module, background tasks module and control module. Every module can provide a diverse set of capabilities. The main advantages of cloud computing relate to dynamic scalable mechanism, parallelism computing and distributed structure.

From Fig.2, we see that the cloud computing enables desktop-based computing to move towards full web-based computing where a web browser can be used to access, develop and configure various applications, hardware and data over the internet. It also indicates that the combination of cloud computing and robotics is an inevitable trend. Many major companies participate in cloud computing and establish all kinds of platforms such as Google App engine [8], Amazon Elastic Compute Cloud (EC2) [9], Microsoft Azure [10], GRIDS Lab Aneka [11] and Sun Grid [12]. The emergence of cloud computing and the corresponding platforms make it possible to conveniently use cheap computing resources in a similar manner to water or electricity in daily life.

The concept of cloud robotics can be traced back at least two decades to network robots. Masayuki Inaba [13] proposed a robot control method based on remote computing, highlighting the advantages and the IEEE Robotics and Automation Society established a technical committee for Networked Robotics in 2001 [14]. Kamei et al., [15] proposed "cloud networked robotics" to fulfil various location-based tasks in a shopping mall for supporting daily activity, especially for the elderly and disabled. The ASORO lab in Singapore proposed DAvinCi based on Hadoop and ROS (Robot Operating System), which shows the scalability and parallelism advantages of cloud computing for service robots in large environments [16]. Tenorth [17] designed the UNR-PF to realize human-computer interaction in a convenience store and Gostainet [18] established an infrastructure using cloud robotics for speech recognition on the humanoid robot NAO. Carlos and Du Z et al. [19] present an architecture design of "robot cloud" to bridge the power of robotics and cloud computing. They use the SOA (Service-oriented architecture) to expand the capacities of physical robots. Nan Tian et al. [20] described Berkeley Robotics and Automation as a Service, which is a RAaaS prototype that allows robots to access a remote server that hosts a robust grasp planning system (Dex-Net 1.0). The above research mainly focuses on some practical application areas but does not present systematic architectures for cloud robotics which is the focus of this paper.

In addition, the European Union also started a groundbreaking cloud robotics project "RoboEarth" [21] in 2009. This project attempts to build a giant network and database repository where robots can share information and knowledge and learn from each other about their behaviors and environments. The researchers have developed the famous cloud engine "Rapyuta" [22] and the knowledge processing system "KnowRob" [23] successfully. Furthermore, in 2014 scientists from institutes including Cornell, Stanford, Google, and Microsoft developed a new project "RoboBrain" [24] that allows robots to learn and share representations of knowledge. These developments indicated that cloud robotics can be used to effectively and efficiently expand the robots' knowledge and skills. Undoubtedly, there is much more additional research being undertaken in the field of cloud robotics. However, from this brief review, we can determine the main cloud robot architectures can be divided into two subgroups: M2M/M2C and UNR-PF and the these are typical distributed monomer architectures. The deployment of a typical distributed monolithic architecture is depicted in Fig.3.

However, the traditional distributed monolithic architecture has also some inherent defects. From the development methodology, many companies want to deploy more applications to the cloud and they also need to innovate

as fast as possible to avoid competition [25]. Therefore, continuous delivery is very important for many startups or large Internet corporates in recent years. Applications based on a typical monolithic architecture would have a single codebase shared among multiple developers and be developed using an MVC (Model View Controller) [26] web application framework such as JEE, .NET, Symfony, Rails, Grails and many others. If these developers want to add or change services, they must do more work to make sure that the new service is perfectly compatible with other services. Thus, as more services are added, the complexity of deployment will increase significantly and limit the ability of companies to innovate with new application versions and features in the monolithic applications. The above limitations and problems have become a great challenge for most Internet companies and SaaS providers which we aim to address in this paper.

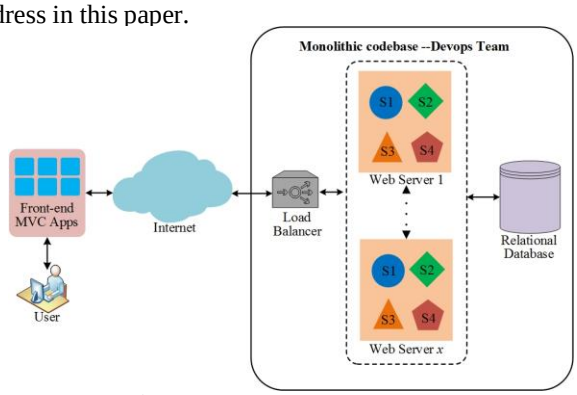


Fig.3 Deployment of the monolithic architecture

B. Microservice in the cloud

To solve the above problem of deployment, we propose a novel lightweight cloud robotics architecture based on microservice. Microservice [27] is a software architecture style in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs. It should be noted that, although the design and philosophy behind each architecture approach share some traits, microservices and the SOA are fundamentally different in other key ways. With the new architecture, the companies can innovate quickly and reduce complexity by using computing resources efficiently. Therefore, the development teams can be enlarged in a controlled way. The brief deployment of the microservice architecture on a cloud solution is depicted in Fig.4. In Fig.4, μS1 , μS2, μS3 and μS4 all are microservices, each of which can be developed using different technological stacks as three tiers applications. The gateway is developed as a light web application that receives requests from end-users and gets or returns the results. It does not contain a persistence layer because no information needs to be stored. Moreover, it must use the services offered by the microservices (μS1 , μS2, μS3 and μS4) through REST (Representational State Transfer). JSON is used as the interchange message protocol between the display module and the gateway, and the gateway and each microservice. From Fig.4, we can see that the gateway and each microservice can be developed and maintained by independent teams as self-managed applications, which facilitates the increase of the number of developers in a more scalable way than is currently available. Besides, we also find that each microservice may be developed with different programming languages such as Python, Java, .NET, PHP, Ruby, etc. The increasing adoption of microservices in the cloud is motivated by the ease of deploying and updating the applications, as well as the provisioned loose coupling provided by dynamic service discovery and binding. Furthermore, structuring the software deployed in the cloud environment using a collection of microservices allows cloud service providers to offer higher scalability guarantees through more efficient utilization of cloud resources, and to restructure the software to accommodate growing consumers' demand dynamically and quickly [28]. The microservice framework attempts to simplify the process of defining service descriptions to promote automatic service consumption in the semantic web. In the framework, the description task can be improved by enabling reusability across service descriptions.

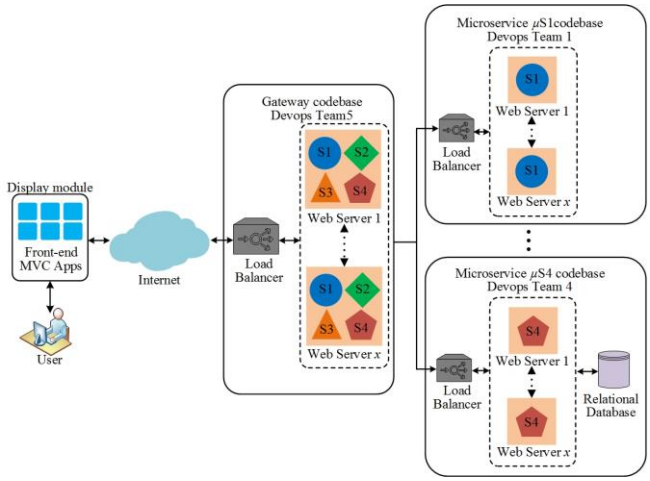


Fig.4 Deployment of the microservice architecture

Recent advancements in the container technology [29] and its capability to overcome limitations in virtualization have shown the advantage of the utilization of containers in the cloud for software applications development and deployment. The container technology is very attractive because it completely enables isolation of independent software applications running in a shared environment. Docker is a representative product of the container technology [30]. The Docker provides a single and lightweight API to manage the execution of containers and allows developers to pre-package the software dependencies into a lightweight and portable file that requires less operation costs than a standard hypervisor. The diagram of microservice based on the container technology is depicted in Fig.5. Microservice supports the realization of small (sized) software applications that are fine-grained and loosely coupled via the REST communication. These applications are implemented using APIs provided by the infrastructure-as-a-service (IaaS) layer for provisioning data computing, storage and delivery capabilities. Besides, the microservice model also enables a simpler and faster migration of software component instances from one visual Machine to another to satisfy variable resource demands for cloud applications.

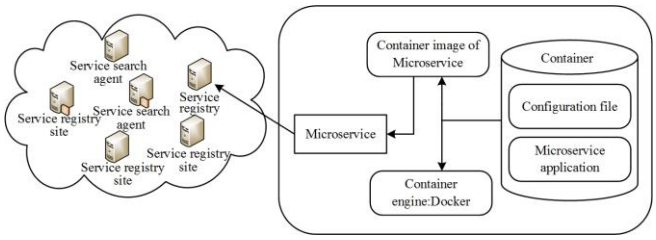


Fig.5 The publishing principle of container-based microservices

III. CLOUD ROBOTICS SYSTEM ARCHITECTURE BASED ON MICROSERVICE USED IN INTELLIGENT SPACE

The motivation of this research is to design a new cloud robotics system framework based on microservice to implement the Cloud-based Assisted Living Project (CALP). From Section II, we find that current research focusses on two main parts: cloud platforms and robots. However, various sensors and monitoring systems in the environment are changing our life substantially, and we cannot ignore the revolution brought about by the Internet of Things for intelligent robots. There is no doubt that the environment around the robots should be regarded as an integral part of the whole CR system and it also plays a significant role in the real application of the CR system. Therefore, in this paper, the CR system mainly includes three parts: cloud platform, robots and robots' working environment. For the CALP, the robot's working environment can be seen as an intelligent environment. The intelligent space (iSpace) [31], also called a smart space or intelligent environment, is a space with devices, multi-source information and communication technologies creating interactive environments that bring computation into the physical world and enhance the occupants experiences; the iSpace is depicted in Fig.6.

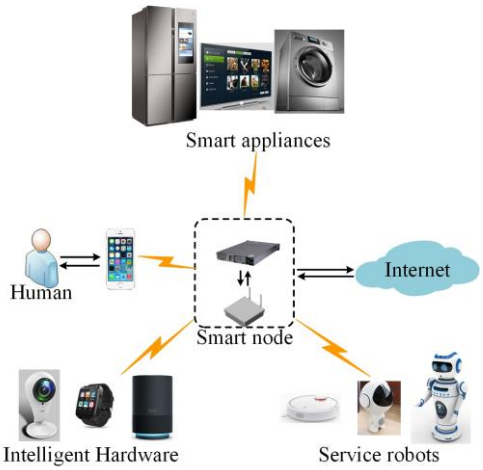


Fig.6 Intelligent space (iSpace). The iSpace can physically and mentally support people through robot and intelligent hardwires, thereby providing satisfaction for their needs.

For cloud robotics system, most of services requested by local physical robots are apparently compute-intensive tasks in the cloud such as SLAM, navigation and scene recognition and fusion. The distributed monolithic architecture often deploys and runs an integrated development application and provides the robot services by the entire system. However, the change of one function may affect the others and cause more difficulties of redeployment and continuous integration due to the change and evolution of the system function. Additionally, since the monomer system adopts a unified technology stack and development standard, it will make the development process more limited and complicated. The discussion indicates that current distributed monomer architectures need to be improved and changed. Based on this, we replace the microservice architecture as a solution. The microservice, as a new software architecture design pattern, has shown the competitive strengths such as more productive, flexible and low development costs. Obviously, it is a very complex work to design a novel architecture based on microservice for the CR system. We need to consider many factors including reliability, scalability, modularity, interoperability, interface and QoS. The core designs of the proposed system are described briefly as follows.

A. Cloud robotics system based on microservice in iSpace

The structure diagram of the proposed CR system based on microservice is depicted in Fig.7. The fundamental idea is that the service architecture can be divided into smaller granularity services that run in an isolated environment. Fig.7 clearly shows the basic components and system composition of cloud robotics that are migrated to the microservice architecture from a single distributed application. In Fig.7, the microservice application is released to the distributed environment via a continuous delivery platform after deployment and verification; then it will be registered. Besides, physical robots can upload the collected multisource data information and request "robot services" towards cloud management system via wireless WIFI and wired short-range network. The iSpace can share all sorts of environment information with the cloud platform. To implement the proposed microservice architecture successfully, these components will be put into a container and be managed by the Docker engine. In addition, the APIs' service will be accessed by the users or external services via Service Gateway. The key components of the proposed architecture are described below in detail.

(1) Service registry and discovery component

Service registration and discovery is the core component of the proposed architecture. In the distributed environment, the service instance will be changed dynamically according to the default rule or policy in a dynamic environment. leading to a higher requirement for this component. The sketch of the service registry and discovery mechanism is depicted in Fig.8. It should be noted that the proposed services refer to "robot services", such as SLAM service, navigation service and vision recognition service.

a) Registration and identification service

Since a microservice application can be deployed via the Continuous Delivery Platform (CDP), it will be registered as

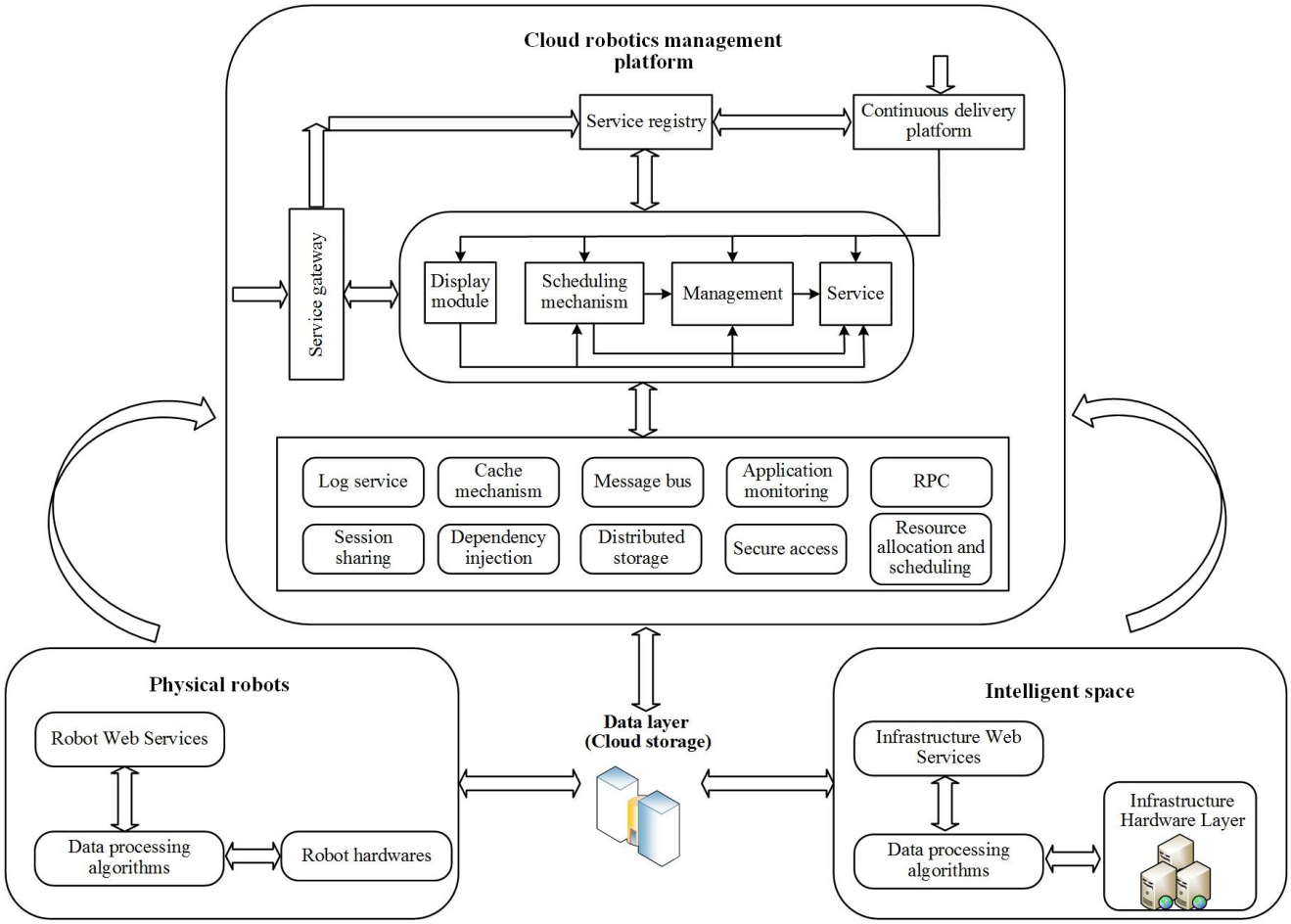


Fig.7 Microservice-based cloud robotics (MCR) system for intelligent space consists of cloud platform, robots and intelligent space. Each function of the MCR system will be seen as a microservice, especially for robot services. The data center shared in this system is built on the cloud servers.

a service instance by the service registry automatically. Besides, the location of service instance will change when the health status and the network environment change, so the service registry needs to track and identify the service instance.

b) Locating and discovery service

Ideally, when the user accesses directly from the client, the scheduling module will query the service registry to find the accessible service and send it to the corresponding service instance via the load balancing algorithms. Dynamic discovery means that calling components can locate microservice information as needed without closely integrating the service. However, an application often relies on the collaboration of several microservices in the real environment, especially for a robot application. For example, SLAM is a complex robot application, that contains tracking service, local mapping service and loop closing service. Therefore, it is very important to locate and discover services. If the caller accesses a service layer directly, it can query the service registration center and find the access service and the corresponding service instances, and then use the load balancing mechanism to invoke the service instance.

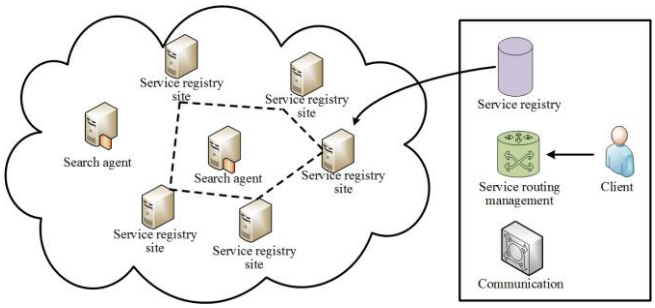


Fig.8 Service registry with non-center nodes and auto discovery mechanism

(2) Sustainable delivery platform

The main function of the sustainable delivery platform is the rapid and flexible deployment of the microservice application. In addition, the deployed microservice must be programmable, easier-to-maintain and scalable, which can run in a separate and isolated container as a process. This sustainable delivery process can bring more rapid feedback to the application. Furthermore, compared with the traditional "Waterfall" software development process, sustainable delivery will become more cooperative and more efficient on demand analysis, user experience, interactive design, testing and maintenance collaboration.

For example, as an important part of the robot application "SLAM", the loop closing often needs to be improved or updated by researchers. If we take full advantage of this platform to publish a new improved version, we believe that the development of SLAM will be easier and faster than what currently exists. The sustainable delivery (deployment) process is depicted in Fig.9.

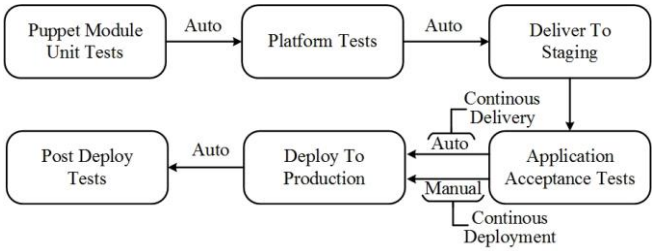


Fig 9. The sustainable delivery (deployment) process

(3) Service gateway

The service gateway (SG) is a unified call logic portal, which encapsulates the service information of a node in a distributed environment. The main functions of the service gateway are described as follows: a) The services which are registered via an existing service registry are exposed to an external call directly; b) The SG can satisfy the requirement that a client requests multiple services at one time; c) Support cache storage for some services whose operating results are constant in certain time intervals. If the service request fails, the SG will provide the last correct cache execution or null response; d) Provide request distribution routing, load balancing, security protection, protocol conversion and other functions.

(4) Log service, application monitoring and RPC

The log service component will accurately collect various pieces of information, such as operation log, SQL operation log, exception log, etc. Then, after standardization, filtering, merging and alarm analysis, it centralizes storage and management in a unified format, helping users to locate faults quickly, and providing objective basis for tracking and recovery by summarizing and analyzing all log information. The monitoring component provides the running status of the microservice, the JVM performance index, the system performance index and the monitoring function of the microservice call chain to facilitate real-time monitoring for users. The RPC (Remote procedure call) component provides a remote procedure call mechanism that is suitable for a distributed environment to ensure the performance and reliability of inter-service communication.

(5) Communication protocol

For the communication pattern, all components can contact each other, no matter how they communicate at the interface or protocol levels. In this paper, we adopt the REST (Representational state transfer) protocol [32] as the communication specification among different microservices and JSON (JavaScript Object Notation) [33] is used as the data format. Compared with the traditional protocol such as the SOAP [34] protocol and WSDL [35], REST and JSON are all lightweight protocols and communicate directly by HTTP requests. The REST is an architecture style that can allow web services to provide interoperability between computer systems via an Internet. The web services based on REST allow requesting systems to access and manipulate text descriptions of various web resources using a unified and predefined set of stateless operations. Through the stateless protocol and the standard operations, REST systems can provide high availability, fast performance, continuous growth capacity and reliability by reusing components that can be managed and updated without impacting the system, even while it is running. JSON is a lightweight data exchange format that uses human-readable and easy-to-edit texts to transmit the data objects containing attribute-value pairs. Moreover, the JSON is also the most common data format that is used for asynchronous browser/server communication. In addition, each service data will be clearly defined in two formats: normal and abnormal. To make the data processing and reading easier, the exception code and exception information of the abnormal format are necessary. The communication flow between different microservices is described in Fig.10.

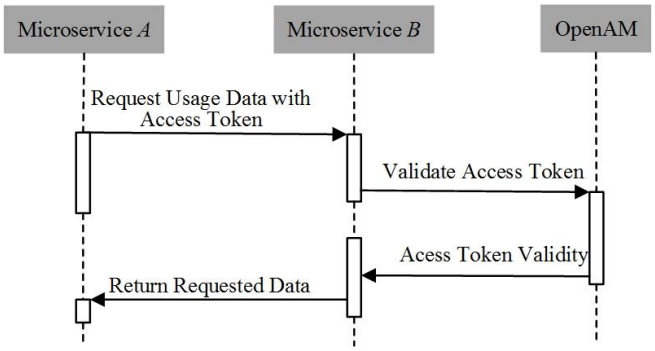


Fig.10 Communication flow between different microservices

When we adopt the instance independence patterns, the proposed architecture can support component-to-component communications by synchronous or asynchronous models. And it does not force the other components to be in any specific state before receiving the requests or messages. Thus, if our proposed deployment is appropriate, all of the services can respond to any requests from components asynchronously and retain or manage every state no matter what the sequence is.

(6) Security mechanism

The data sharing mechanism of our proposed architecture supports the cross platform and the cross application. It is very important to ensure safe and reliable access while unauthorized access is denied. The flow chart of security mechanism based on OAuth2 protocol is depicted in Fig.11. To enable the security mechanism, we will take advantage of the federated security system that can create trust between components, no matter whether the security model is local to the components. OAuth is an open protocol that provides a safe, open and simple standard API service for the authorization of user resources without providing the passwords or keys. Moreover, any third party can use the OAuth authentication service and any service providers can their own OAuth authorization service. The security mechanism has been widely used by many Internet companies such as Google, Facebook, Microsoft and Twitter to permit users to share their account information with third party applications or websites. Therefore, we use OAuth

protocol as every microservice's authorization standard in our proposed system.

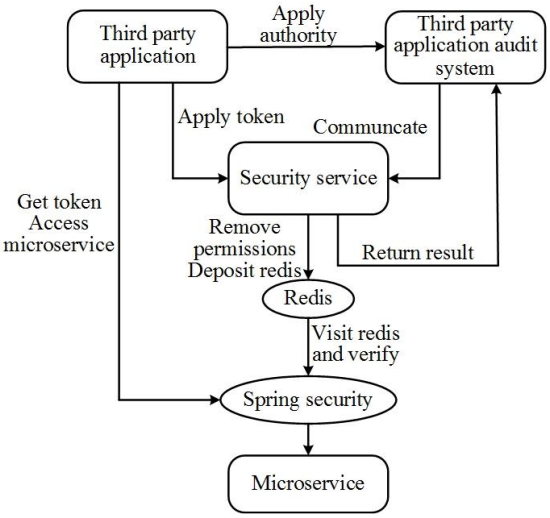


Fig.11 The flowchart of security mechanism

B. Scheme demonstration and feasibility analysis

The proposed scheme in this paper comes with some irreplaceable advantages, including the ability to reduce complexity by leveraging container abstractions which indicates that we can abstract the access to resources (such as storage) and make the application portable thus speeding up the refactoring of the applications by removing dependencies on the underlying infrastructure services. In the past, most of researches focusing on cloud robotics architecture, security and governance services have been platform specific, not application specific. It is obvious that the traditional on-premises applications have almost no security and governance functions. Therefore, the new proposed architecture can provide better portability and less complexity by placing security and governance services outside of the application domain. Moreover, in the proposed architecture, the applications can be distributed and optimized according to their utilization of the platform. When the proposed architecture is adopted, we can easily place an I/O-intensive portion of the application on the cloud thus providing better performance than non-cloud based approaches, placing a computationally-intensive portion of the application on a public cloud that can provide the proper scaling and load balancing. Additionally, an important prerequisite for the proposed framework is that all of these elements work together to form the application and the applications should be divided into components that can be optimized. It means that the application should be broken down to its functional primitives and built it up as component pieces to minimize the amount of code that needs to be revised.

From an operational point-of-view, the nature of the operations in the proposed CR system based on microservice is cloud operations [36], or the operation of the application containers in the cloud. Before the applications are generated, developers should take advantage of the microservice architecture and container technology. They should manage the applications as distributed components that can be separately scaled. For example, the container that manages the user interface can be easily replicated in servers when the demand increases within a certain time period. This indicates that the operation is a very convenient way to achieve the scalability automatically around the application thus expanding the use of cloud resources as needs change. Although the proposed method forces the application developers to think about how to best redesign the applications to make them containerized and service-oriented, it is more productive, flexible and cost effective. Compared with the complexity of the cloud operations, the advantages of the follow-up use also verify the feasibility and necessity of the proposed architecture.

Additionally, two important considerations regarding the proposed system are programmability and ease of use. Programmability can be regarded as a set of tools and best practices to add, deploy and manage applications' microservices. Although the learning process is necessary, compared with other traditional CR systems the proposed system required lower learning threshold and less learning cost. According to our research, if you have a general familiarity with new applications and docker technology, you can develop and deploy the applications in the proposed system. Ease of use is also a fundamental consideration factor for developers and users. For the proposed system, microservice allows to develop and maintain the applications in different programming languages, which is unimaginable for traditional CR systems. Moreover, microservice can make the deployed applications clearer and more convenient for users. These advantages make the proposed system architecture more competitive than other CR systems.

IV. Experiment And Analysis

We use Kubernetes (commonly referred to as "K8s") [37] to build a service cluster environment and adopt the Docker engine to implement service encapsulation. In addition, the cloud platform uses our own private cloud which is built by OpenStack [38].

A. Experimental system deployment

To complete the deployment of the cloud management development environment, we create four nodes: one master node, two slave nodes and one docker private node. The private node provides mirroring services for a cluster environment as a private warehouse server of Docker. The details of all nodes are shown in Table.1. Then, we need to install the Flannel service and configure the environment of Kubernetes. We also change the configuration file of Docker to install some components for the master node such as the API server, Scheduler and Controller Management, and install other components for slave nodes such as Kubelet and Kubernetes Proxy. Obviously, to ensure the components work, we should create the Service configuration file to manage and control the system [39].

In addition, we choose the new ORB-SLAM2 [40] as an application of FaaS (Function as a service) in a cloud management platform. The ORB-SLAM2 can work in real time on the standard CPUs (Central processing units) in a wide variety of environments from small hand-held indoors

scene to drones flying in industrial environments and cars driving around a city for monocular, stereo and RGB-D cameras. Due to the complicated structure and intensity of technologies, SLAM is very difficult to modify or develop for most of scholars and researchers. Therefore, we split the ORB-SLAM2 process into three microservices: the tracking service, the local mapping service and the loop closing service, as is depicted in Fig.12. These microservices are built in a shared database. Then we define three related Dockerfiles and use the command "$ sudo docker build ~" to construct Docker images. Finally, we publish the docker images on a docker private mirror server. Besides, it should be noted that the docker mirror name must be prefixed with an IP number of the Docker private server, such as "19216859131".

When the experimental system environment has been deployed, we need to create a configuration file by YAML to build resources and start services. Therefore, we should create three Replication Controller (RC) files and the related Kubernetes Service files. The RC file and Service file in the pose map are shown below.

Then, we can start the service using the command "$ kubectl ~" on the master node. Moreover, it should be noted that every microservice is assigned a cluster-IP address, which is an access entry address defined by Kubernetes. Using the IP address, we can access the cluster instances consisted of Pod copies.

Table.1 The details of all nodes

Nodes	Node IP	Operation System	Configuration
Master Node	192.168.59.128	CentOS 7	Memory: 2G; Storage: 10G
Slave Node 1	192.168.59.129	CentOS 7	Memory: 16G; Storage: 20G
Slave Node 2	192.168.59.130	CentOS 7	Memory: 16G; Storage: 20G
Docker private server	192.168.59.132	Ubuntu 14	Memory: 2G; Storage: 40G

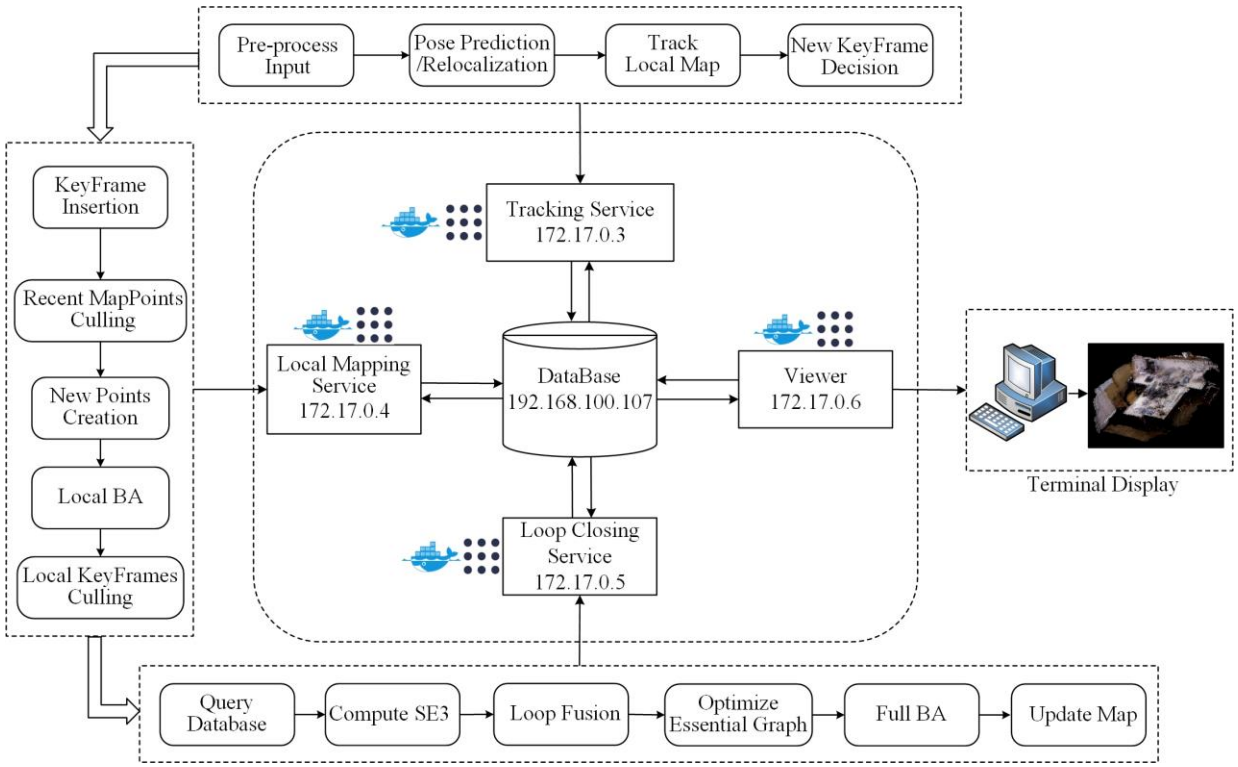


Fig.12 The ORB-SLAM2 is divided into three nodes (microservice): tracking, local maping and loop closing. Each microservice can be assigned a independent IP address and deployed in a separate container. The case builds the communication between nodes by ROS.

1) RC file:

```
apiVersion: v1
kind: ReplicationController
metadata:
    name: pose
spec:
    replicas: 2
    selector:
        app: pose
    template:
        metadata:
            labels:
                app: pose
        spec:
            containers:
            - name: pose
              image: 192.168.59.132:5000/pose
              ports:
              - containerPort: 3306

```

2) Service file:

```
apiVersion: v1
kind: Service
metadata:
    name: pose
spec:
    ports:
        - port: 3306
    selector:
        app: pose

```

B. Simulation using the standard dataset

To verify the proposed system, we use the standard dataset "freiburg2-desk" [41] from TUM for simulation testing. The tested dataset information is described in Table 2. The point cloud map and octomap built by the microservice-based ORB-SLAM2 are depicted in Fig.13. The comparison of estimated trajectory and ground truth is depicted in Fig.14. From Fig.13, we can find that the microservice-based ORB SLAM2 (M-ORB-SLAM2) system runs well and the point cloud map and octomap are all sufficiently clear for the robot's navigation and relocation. Fig.14 shows that the error of the M-ORB-SLAM2 is relatively small and meets the necessary level of accuracy.

Then we use the Root Mean Square Error (RMSE) of the Absolute Trajectory Error (ATE) and running time to accurately evaluate the system performance. Suppose the estimated robot pose is $\hat{X} = \left\{ \hat{X}_1, \cdots, \hat{X}_n \right\}$, and the real moving trail is $X = \left\{ X_1, \cdots, X_n \right\}$. Then the RMSE of the ATE can be calculated as follows:

$$ATE_{RMSE}(\hat{X}, X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[trans(\hat{X}_i) - trans(X_i) \right]^2} \quad (1)$$

where *trans* is the translation vector.

To strengthen the argument, we compared the system with the LSD-SLAM, RGBD-SLAM, ORB-SLAM2. Every method is run 10 times and the average results are used as an evaluation index for comparison and shown in Table 3. From Table 3, we can see that the differences of the ATE and RMSE between ORB-SLAM2 and the proposed method are all less than 0.001m. The difference of running time is less than 2s. The results indicate that the M-ORB-SLAM2 has similar performance to the ORB-SLAM2. Additionally, compared with the ORB-SLAM2, the M-ORB-SLAM2 system tends to be more loosely coupled, heterogeneous and physically dispersed and each component can be easy to be modified or developed by researchers and technicians. It indicates that the proposed method will encourage more people to participate in improving a particular component of SLAM without knowledge of other components. Moreover, this also indicates that the M-ORB-SLAM2 is extensible and can be reused by other teams, ultimately promoting standardization. Based on the above, the M-ORB-SLAM2 can perform better than LSD-SLAM and RGBD-SLAM.

Table.2 The tested dataset basic information

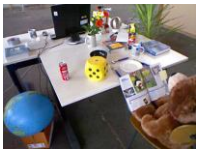
Samples of frames	Basic information
   	Distance:18.880m Time: 99.36s Mean angular velocity: 6.338deg/s Mean linear velocity: 0.193deg/s Number of frames: 2893 Trajectory size: 3.90m×4.13m×0.57m

Table.3 Comparison of SLAM system performance

	LSD-SLAM	RGBD-SLAM	ORB-SLAM2	M-ORB-SLAM2
Average error /m	0.038	0.090	0.010	0.009
RMSE /m	0.045	0.095	0.011	0.012
Running time /s	>500	>500	271.5	269.7

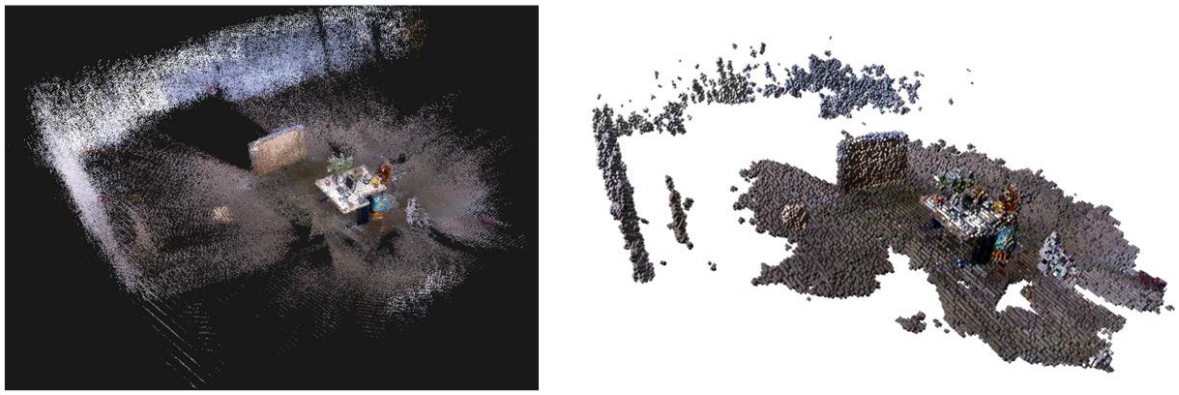
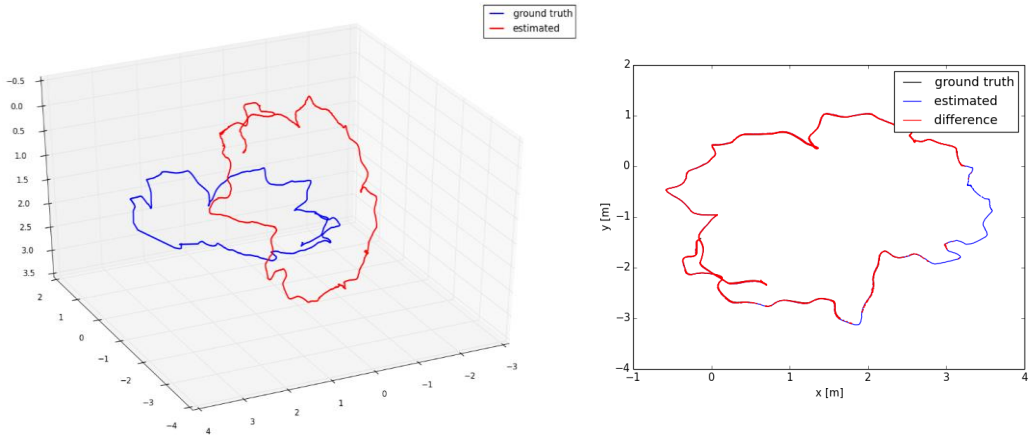


Fig.13 The simulation results of "freiburg2-desk" based on the proposed method: (a) dense point cloud map; (b) octomap map



(a) (b)

Fig.14 Comparison of estimated trajectory and ground truth: (a) 3D visual angle; (b) 2D projection angle

C. Experiment in a real-world scenario

To further verify the proposed M-CR system, we design a SLAM experiment in a real scenario. We use the above M-ORB-SLAM2 as the mapping method in this experiment. The tested scene is an indoor study room of our college. To simulate the environment of the CALP, we created a simple smart space with some Ultra-Wideband (UWB) modules. These UWB modules can assist 3D map building and localization in the work space. The main experimental hardware equipment is a TurtleBot with mounted Kinect1.0 and the embedded board NXP.I.MX6Q. The experimental indoor floor plan is depicted in Fig.15. Detailed information of all required hardware equipment is described in Table 4.

In conclusion, the complete flow chart of the experiment is described in Fig.16. It is important to note that the embedded processor in this experiment is low cost and unable to complete the whole visual SLAM process alone. In addition, the communication mechanism between cloud and robot adopts Socket. The communication method between physical robot and cloud mainly uses the wireless WIFI. From Fig.16, we can also see that the cloud management platform is the key part for the proposed Microservice-CR system. To investigate the accuracy of the mapping and location service in smart space, we imitated the smart home and designed a model of a simple realistic home scenario in our laboratory. The labels "A" to "E" represent desks with sensors. The labels "G" to "I" represent chairs with sensors. The label "F" represents the door with a sensor. In this experiment, the robot needs to fulfil two tasks: build a 3D map of the tested space and locate these desks, chairs and the door.

Table.4 Required hardware equipment

Equipment	Detailed configuration	No.
Private Cloud Servers	Intel Xeon E5-2620v4 CPU×2, 2.4GHz, 64G DDR4ECC memory, 2T storage, GTX1080, 4xPCIE3.0x16	4
Robot	TurtleBot platform	1
Embedded processor	NXP(Freescale) I.MX6Q Cortex A9 CPU, 4cores@1.2GHz, 2G RAM, 16G eMMC Flash	1
Visual sensor	Kinect1.0 from Microsoft	1
PC	Intel Core i5, 8GB DDR4 memory, 256GB SSD storage, NVIDIA 940MX monitor,	1
Other sensors	UWB modules	9-10

Task 1:3D visual SLAM task

Though the proposed architecture is very lightweight and easily expandable, it can also easily obtain huge computing resource and tackle all kinds of computation tasks due to cloud platform. The 3D mapping result is described in Fig.17.

Since 3D visual mapping is a typical computationally intensive task, we use the running time as a performance index to evaluate the system. To make the experiments more convincing and appealing, we use "Running locally by laptop" and the monomer CR system as comparative schemes. For all schemes, the key frames processed by embedded-level robots will be sent to receivers (laptop or cloud servers) using wireless transmission technology such as WIFI. It should be noted that the laptop used for the first comparative scheme has the following general hardware configurations: Core i5@2.5GHz, 4G memory and 1T Storage. Moreover, the monomer CR system is a non-microservice CR system based on Robot Operating System (ROS) [42], which implies that SLAM in a cloud platform is a whole package and the robot is just regarded as a simple image acquiring unit. The communication protocol of the monomer CR system based on ROS is Rosbridge. The detailed development or deployment information of the three schemes is described in Table 5. The comparison of the run-times of the whole 3D visual SLAM is depicted in Fig.18.

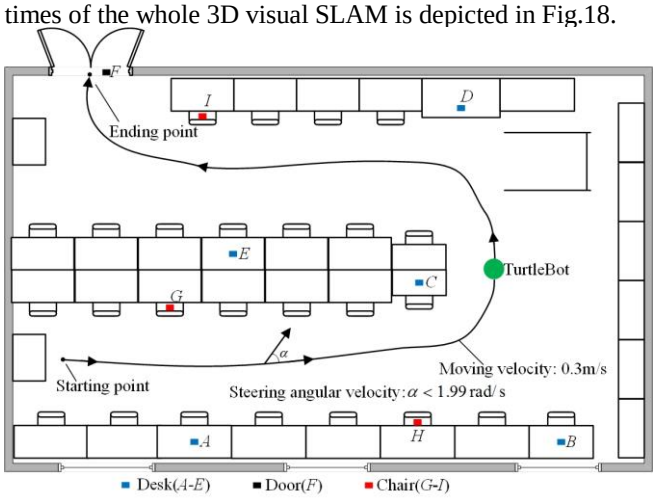


Fig.15 The diagram of the tested smart space

From Table 5, we can see that the proposed system has almost the similar deployment time with scheme 2. However, the proposed system has better scalability and computing capability than others. Besides, compared with other schemes the proposed system can offer more diverse and friendlier development languages support such as Java, Python, C# and Ruby. Undoubtedly, the advantage reduces the development threshold and allows more people to participate in the improvement of robot applications and services. Moreover, Table 5 indicates that the proposed system can offer easier code maintenance, which can reduce the cost of operation and maintenance. The low hardware cost of the proposed system allows more "Shortage of Funds" researchers to do some "expensive" robot research work. Since deployed applications need to be split into "microservices", they have lower coupling degree and more flexible deployment. Moreover, the proposed system architecture can build and improve the SLAM process more flexibly than other existing architectures. For example, we can easily replace the closed-loop module with deep convolutional neural network to improve the mapping accuracy. Additionally, we expediently check and modify every microservice to remain in an optimal state. In Table 5, we make a detailed correlation among these schemes with respect to various aspects such as the deployment time and language, coupling, scalability and cost. Table 5 shows that the proposed system has relatively more superiority than other schemes. Though the deployment of the proposed system takes longer, it will be acceptable and worthwhile for sustainable development and research.

From Fig.17 we see that the 3D mapping based on a cheap embedded board processor is successful and the reconstructed indoor scene of the laboratory is very clear. The result entirely meets the requirements of robot location and navigation in a real scenario. Fig.18 shows that the difference in the average run time of each key frame among three schemes is less than 8ms. From Fig.18, we see that the proposed architecture has almost the same run time as the other two methods. It also indicates that splitting into microservices do not reduce the execution speed of SLAM process. This is because the communication mechanism among the microservices is like the memory reading and writing process, and these communication time can often be ignored. The results indicate that the proposed architecture is effective and the deployment of the microservice application is successful. Though it is not perfect as a prototype system, there is no doubt that the proposed architecture presents a better solution on cloud robotics than the past research work.

Table 5 Comparison of three schemes in detail

No.	Deployment time(/h)	Development language	Coupling degree	Scalability	Computing capability	Code maintenance difficulty	Communication protocol (Robot-cloud)	Hardware cost (robot controller)
Scheme 1	>6	C++	High	Bad	+	Hard	None	Like a laptop ($800)
Scheme 2	>12	C++	Medium	Normal	+ + +	Medium	Rosbridge	Embedded device (<$50)
Scheme 3	>12	No limit	Very Low	Very good	+ + +	Easy	Socket	Embedded device (<$50)

Note: Scheme 1: Running locally using a laptop level controller; Scheme 2: The monomer CR system based on ROS; Scheme 3: The proposed Microservice-based CR system

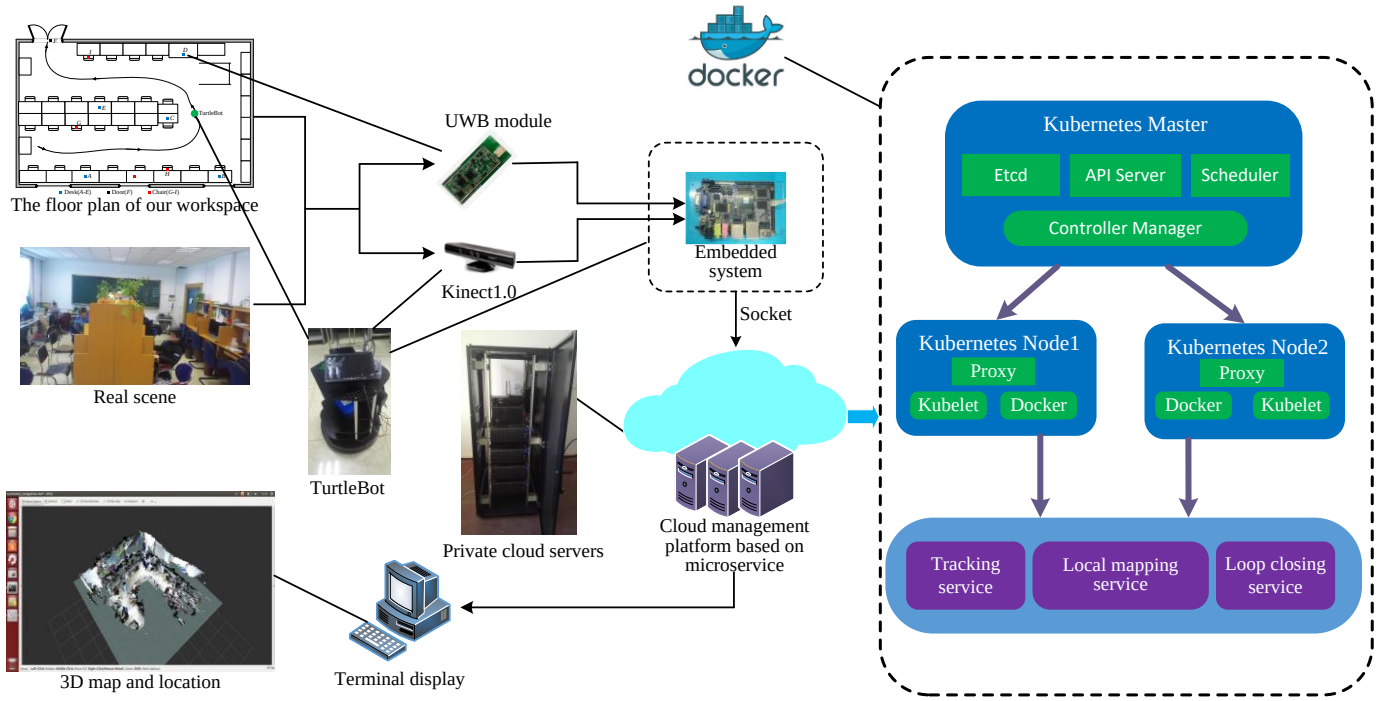


Fig.16 An overview of the entire experimental setup.

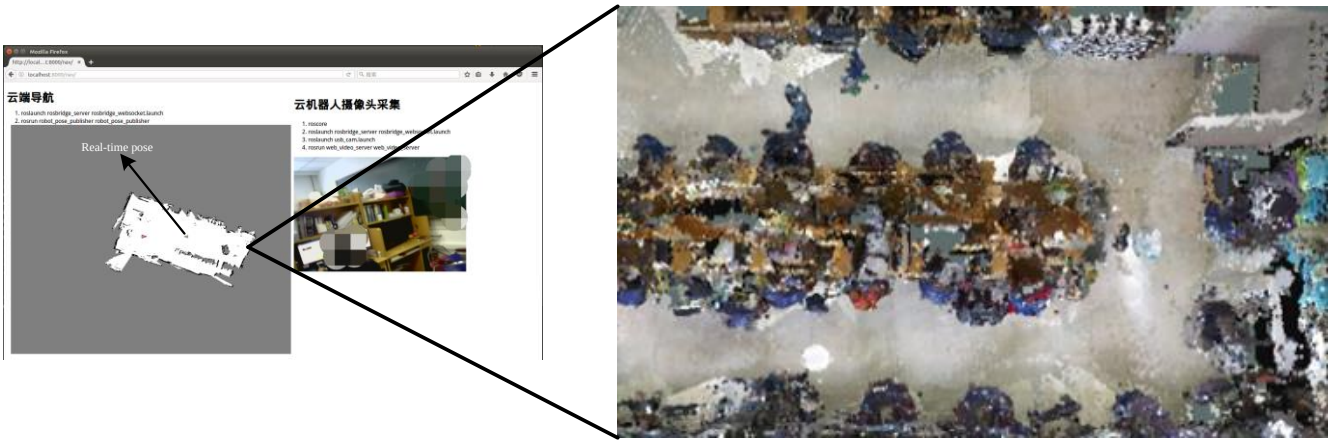


Fig.17 The 3D map of our workspace

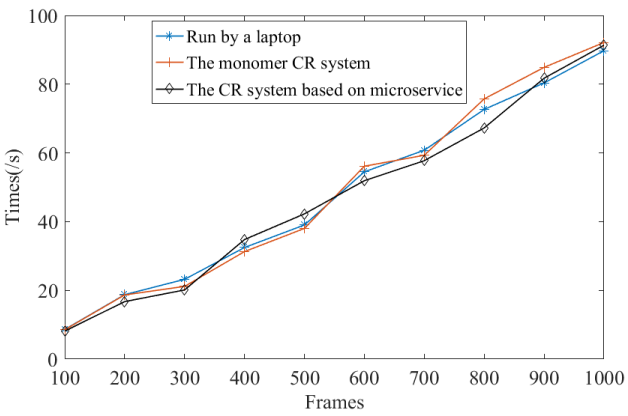


Fig.18 Comparison of running time using different frameworks

Task 2: Location task

When the 3D map has been built, a robot can determine the position of the labels by self-localization and UWB signal. Therefore, we regard the location as a microservice and package the third-party application using Docker to verify the scalability of the proposed architecture. Since different microservices share the same database, the communications consumption in the cloud platform can be ignored. However, it should be noted that the positioning effect depends on map accuracy and the robot's self-localization accuracy. Thus, in this paper we run the location experiment 10 times and use the average as the final result. The location experiment's result is depicted in Table 6; the location precision is 0.1m. In Table 6, the location results show that the robot can judge the location of the desks and

chairs effectively. From the experiment, we find that the third-party application can be developed independently and is easier to deploy for the proposed architecture than a traditional monomer architecture.

Table 6 The location experiment's results

	Label				
Desk	*A*	*B*	*C*	*D*	*E*
Accuracy	0.9	0.8	1.0	0.8	0.9
Chair	*G*	*H*	*I*		
Accuracy	1.0	0.9	0.9		
Door	*F*				
Accuracy	1.0				

V. Conclusion and Future Work

We have presented a new cloud robotics prototype system architecture based on microservice for use in an intelligent environment. We demonstrate the proposed system performance using both a simulation test and a real experiment and this highlights that both run-time and flexibility of the proposed approach are comparable with existing architectures. Additionally, third-party applications can be developed independently and are easy to deploy for the proposed architecture than a traditional monomer architecture. There is no doubt that the CR system based on microservice is a very important and prospective research for the development of intelligent robots. The microservice-based ORB-SLAM2 also presents a meaningful exploration for standardizing the visual SLAM process.

Future development of the research direction will enable proposed system architecture to show its superiority and bring more convenience and benefits. The proposed architecture is currently just a prototype system; we will continue with in-depth study to improve it further. In the future, we hope that the Microservice-CR system can play an important role in the field of home service robots, especially for the disable and the elderly.

Acknowledgment

We would like to thank Professor Sonya Coleman and other members from Intelligent System Research Centre (ISRC) of Ulster University for helping us to improve this work. Additionally, the authors would like to thank experienced anonymous reviewers for their constructive and valuable suggestions for improving the overall quality of this paper.

References

[1] Kuffner J J. Cloud-enabled robots[C]//IEEE-RAS international conference on humanoid robotics, Nashville, TN. 2010.

[2] Kehoe B, Patil S, Abbeel P, et al. A survey of research on cloud robotics and automation [J]. IEEE Transactions on Automation Science and Engineering, 2015, 12(2): 398-409.

[3] Villamizar M, Garcés O, Castro H, et al. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud[C]// 10th Computing Colombian Conference (10CCC), IEEE, 2015: 583-590.

[4] Lorido-Botran T, Miguel-Alonso J, Lozano J A. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments[J]. Journal of Grid Computing, 2014, 12(4): 559-592.

[5] Hu G, Tay W P, Wen Y. Cloud robotics: architecture, challenges and applications [J]. IEEE network, 2012, 26(3): 21-27.

[6] Kamei K, Nishio S, Hagita N, et al. Cloud networked robotics[J]. IEEE Network, 2012, 26(3): 28-34.

[7] Mell P, Grance T. The NIST definition of cloud computing [J]. Communication of the Acm, 2011, 53 (6): 50-50.

[8] Ciurana E. Developing with google app engine[M]. Apress, 2009.

[9] Amazon Elastic Compute Cloud (EC2). http://www.amazon.com/ec2/.

[10] Microsoft Azure. http://www.microsoft.com/windowsazure/

[11] Chu X, Nadiminti K, Jin C, et al. Aneka: Next-generation enterprise grid platform for e-science and e-business applications[C]// IEEE International Conference on e-Science and Grid Computing, 2007: 151-159.

[12] Sun network.com (Sun Grid). http://www.network.com.

[13] Inaba M. Remote-Brained Robots [C]// Multisensor Fusion and Integration for Intelligent Systems, 1994. IEEE International Conference on MFI '94, IEEE, 1997: 747-754.

[14] IEEE networked robots technical committee. [Online]. Available: http://www-users.cs.umn.edu/~isler/tc/

[15] Kamei K, Nishio S, Hagita N, et al. Cloud networked robotics[J]. IEEE Network, 2012, 26(3): 28-34.

[16] Arumugam R, Enti V R, Bingbing L, et al. DAvinCi: A cloud computing framework for service robots[C] // 2010 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2010: 3084-3089.

[17] Tenorth M, Kamei K, Satake S, et al. Building knowledge-enabled cloud robotics applications using the ubiquitous network robot platform[C]// IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2013: 5716-5721.

[18] Mester G. Cloud Robotics Model [J]. Interdisciplinary Description of Complex Systems, 2015, 13(1): 1-8.

[19] Du Z, He L, Chen Y, et al. Robot Cloud: Bridging the power of robotics and cloud computing [J]. Future Generation Computer Systems, 2016, 21(4): 301-312.

[20] Tian N, Matthew M, Mahler J, et al. A cloud robot system using the dexterity network and berkeley robotics and automation as a service (Brass)[C]// IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017: 1615-1622.

[21] Hunziker D, Gajamohan M, Waibel M, et al. Rapyuta: The roboearth cloud engine[C]// IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2013: 438-444.

[22] Tenorth M, Beetz M. KnowRob: A knowledge processing infrastructure for cognition-enabled robots [J]. The International Journal of Robotics Research, 2013, 32(5): 566-590.

[23] Saxena A, Jain A, Sener O, et al. RoboBrain: Large-Scale Knowledge Engine for Robots[J]. Computer Science, 2014.

[24] Waibel M, Beetz M, Civera J, et al. Roboearth [J]. IEEE Robotics & Automation Magazine, 2011, 18(2): 69-82.

[25] Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables DevOps: migration to a cloud-native architecture[J]. IEEE Software, 2016, 33(3): 42-52.

[26] Pop D P, Altar A. Designing an MVC model for rapid web application development[J]. Procedia Engineering, 2014, 69(1): 1172-1179.

[27] Esposito C, Castiglione A, Choo K K R. Challenges in Delivering Software in the Cloud as Microservices [J]. IEEE Cloud Computing, 2016, 3(5): 10-14.

[28] Sill A. The Design and Architecture of Microservices [J]. IEEE Cloud Computing, 2016, 3(5): 76-80.

[29] Stubbs J, Moreira W, Dooley R. Distributed systems of microservices using docker and serfnode [C]// 2015 7th International Workshop on Science Gateways (IWSG), IEEE, 2015: 34-39.

[30] Bernstein D. Containers and cloud: From lxc to docker to kubernetes [J]. IEEE Cloud Computing, 2014, 1(3): 81-84.

[31] Lee J H, Morioka K, Ando N, et al. Cooperation of distributed intelligent sensors in intelligent environment [J]. IEEE/ASME Transactions on Mechatronics, 2004, 9(3): 535-543.

[32] Richardson L, Amundsen M, Amundsen M, et al. RESTful Web APIs [M]. O'Reilly Media, 2013.

[33] Zaidi R. JavaScript Object Notation (JSON)[M]// JavaScript Essentials for SAP ABAP Developers. Apress, 2017.
[34] Dumusque X, Boisse I, Santos N C. SOAP 2.0: A tool to estimate the photometric and radial velocity variations induced by stellar spots and plages [J]. The Astrophysical Journal, 2014, 796(2): 132.
[35] Zheng Z, Zhang Y, Lyu M R. Investigating QoS of real-world web services [J]. IEEE Transactions on Services Computing, 2014, 7(1): 32-39.
[36] Linthicum D S. Practical Use of Microservices in Moving Workloads to the Cloud[J]. IEEE Cloud Computing, 2016, 3(5): 6-9.
[37] Brewer E A. Kubernetes and the path to cloud native[C]// ACM Symposium on Cloud Computing, ACM, 2015:167-167.
[38] Rosado T, Bernardino J. An overview of openstack architecture[C]//Proceedings of the 18th International Database Engineering & Applications Symposium. ACM, 2014: 366-367.
[39] Sefraoui O, Aissaoui M, Eleuldj M. OpenStack: Toward an Open-source Solution for Cloud Computing[J]. International Journal of Computer Applications, 2012, 55(3):38-42.
[40] Mur-Artal R, Tardos J D. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras[J]. IEEE Transactions on Robotics, 2016, 33(5):1255-1262.
[41] Freiburg2_desk dataset. https://vision.in.tum.de/data/datasets/rgbd-dataset/download
[42] Quigley M, Conley K, Gerkey B, et al. ROS: an open-source Robot Operating System[C]//ICRA workshop on open source software. 2009, 3(3.2): 5.